ITCS 209 Object Oriented Programming	Name:	Lab Score	Challenge Bonus
	ID:		
	Section:		

Lab 10: Interfaces and Abstract Classes

Abstraction is an essential concept in OOP that allows client code to make assumptions about how to interact with classes without knowing their actual implementation. In Java, some of the mechanisms you can use to achieve abstraction are ***abstract classes*** and ***interfaces***.

In this lab assignment, an abstract class `Object3D` and interface `Meltable` are available for you. Do not modify Object3D.java except where indicated so (i.e., "You Code Here"). Do not modify Meltable.java.

Object3D provides variables, methods, and abstract methods for an object in 3D, whose common properties include name, material, volume, surface area, potential energy from dropping, the ability to float on water, and the ability to fly in the air.

Different materials have different densities. The following materials and corresponding densities are used in this question and also defined as enum `Material` and array `DENSITIES` in `Object3D`.

Material	Water	Rubber	Gold	OakWood	Butter	Wax	Soap	Air	Oxygen	Hydrogen	Helium
Density (kg/m^3)	1000	1522	19320	760	911	961	801	1.225	1.43	0.08988	0.1664

An object can **float** on water if its material is **less dense than water**. Likewise, an object can **fly** if its material is **less dense than air**.

The calculation of volume and surface area depends on the type of object. Once the volume is known, the mass and dropping potential energy are defined as follows:

$$mass = density \times volume$$

The units of mass, density, and volume are kg, kg/m^3, and m^3, respectively.

Once the ***mass*** is known, one could compute the potential energy (***PE***) at the point of impact from dropping the object from the given ***height*** in vacuum space as follows:

$$PE = mass \times G \times height$$

The units of PE, mass, G, and height are joule, kg, m/s^2, and m, respectively. `G` is the constant gravitational field, already defined in `Object3D`. Consult `Object3D` for more information about existing constants, enum, class variables, class methods, and abstract methods. You must complete the following tasks.

Task 1 (10 points): Abstract Class

Implement the following additional methods in `Object3D` class.

`public double getMass()`: Calculate and return the mass of this Object3D.

`public double getDroppingPotentialEngery(double height)`: Calculate and return the PE from dropping this `Object3D` from the given `height`.

`public boolean canFloat()`: Return `true` of this `Object3D` can float on water; `false` otherwise.

`public boolean canFly()`: Return `true` of this `Object3D` can fly in the air; `false` otherwise.

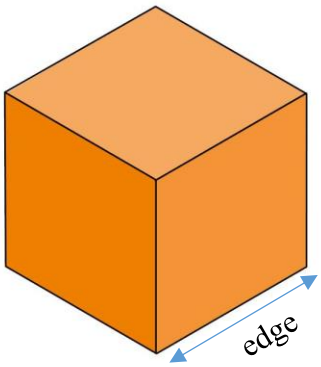
Running `testTask1()` in `StudentTester` should give the following output:

```
[OakWoodBlob(OakWood), V: 125.00, S: 150.00, M: 95,000.00, PE from 9m Height: 8,379,000.00, CanFloat: true, CanFly: false]
[HydrogenBlob(Hydrogen), V: 125.00, S: 150.00, M: 11.24, PE from 9m Height: 990.93, CanFloat: true, CanFly: true]
[GoldBlob(Gold), V: 125.00, S: 150.00, M: 2,415,000.00, PE from 9m Height: 213,003,000.00, CanFloat: false, CanFly: false]
```

Task 2: Utilizing Abstract Classes

For this task, implement classes `Cube` (in Cube.java) and `Sphere` (in Sphere.java), from scratch, by ***extending*** `Object3D`, with the following additional specification:

Cube: A cube is a three-dimensional solid object bounded by six square faces. The edge length governs the size of a cube. The volume and the surface area of a cube are computed as follows:



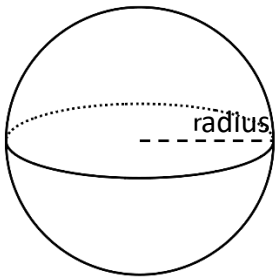
$$volume = edge^3$$
$$surface\ area = 6 \times edge^2$$

Implement the inherited abstract methods, and add the following methods:

`public Cube(String _name, Material _matType, double _edge)`: Constructor that appropriately calls to the super class' constructor and initializes the cube with the given edge length `_edge`.

`public double getEdge()`: Return the edge length of this cube.

Sphere: A sphere is a geometrical object in three-dimensional space that has a ball-like shape. The volume and surface area of a sphere are defined as:



$$volume = \frac{4}{3} \times PI \times radius^3$$
$$surface\ area = 4 \times PI \times radius^2$$

Implement the inherited abstract methods, and add the following methods:

`public Sphere(String _name, Material _matType, double _radius)`: Constructor that appropriately calls to the super class' constructor and initializes the sphere with the given radius `_radius`.

`public double getRadius()`: Return the radius of this sphere.

Running `testTask2()` in `StudentTester` should give the following output:

```
[[RubberCube(Rubber), V: 42.88, S: 73.50, M: 65,255.75, PE from 9m Height: 5,755,557.15, CanFloat: false, CanFly: false],
 [OxygenCube(Oxygen), V: 1,000,000.00, S: 60,000.00, M: 1,430,000.00, PE from 9m Height: 126,126,000.00, CanFloat: true, CanFly: false],
 [HeliumCube(Helium), V: 259.69, S: 244.23, M: 43.21, PE from 9m Height: 3,811.39, CanFloat: true, CanFly: true],
 [GoldSphere(Gold), V: 1,304.84, S: 577.36, M: 25,209,520.68, PE from 9m Height: 2,223,479,724.20, CanFloat: false, CanFly: false],
 [WaxSphere(Wax), V: 4,159,095.41, S: 125,047.97, M: 3,996,890,691.40, PE from 9m Height: 352,525,758,981.48, CanFloat: true, CanFly: false],
 [HydroSphere(Hydrogen), V: 766,714,218,275.03, S: 405,041,691.14, M: 68,912,273,938.56, PE from 9m Height: 6,078,062,561,380.97, CanFloat: true, CanFly: true]]
```

Task 3: Utilizing Interfaces

Implement `WaxDie` (in WaxDie.java) and `ButterBall` (in ButterBall.java) ***from scratch***. *Note that "die" is the singular form of "dice."*

WaxDie:

A `WaxDie` is a cube whose material is wax. `WaxDie` ***extends*** `Cube` and ***implements*** `Comparable<Object3D>` and `Meltable`. Specifically, add a new constructor:

`public WaxDie(String _name, double _edge)`: Call to an appropriate superclass' constructor to set the name and edge of the object, and set material to `Wax`.

Furthermore, implement the following methods from the implemented interfaces:

`public int compareTo(Object3D o)` from `Comparable` to allow ranking the dice by ***volumes*** (ascending). If the volumes are equal, break the tie with the names (lexicographically compared, ascending).

`public Object3D convertToOtherShape()`: Return a `Sphere` with the same name, material, and volume. You can use `Math.cbrt()` to compute cube root (i.e., `Math.cbrt(x)` = $\sqrt[3]{x}$).

ButterBall:

A `ButterBall` is a sphere whose material is butter. `ButterBall` ***extends*** `Sphere` and ***implements*** `Comparable` and `Meltable<Object3D>`. Specifically, add a new constructor:

`public ButterBall(String _name, double _radius)`: Call to an appropriate superclass' constructor to set the name and radius of the object, and set material to `Butter`.

Furthermore, implement the following methods from the implemented interfaces:

`public int compareTo(Object3D o)` from `Comparable` to allow ranking the balls by ***surface areas*** (ascending). If the surface areas are equal, break the tie with the names (lexicographically compared, ascending).

`public Object3D convertToOtherShape()`: Return a `Cube` with the same name, material, and volume. You can use `Math.cbrt()` to compute cube root (i.e., `Math.cbrt(x)` = $\sqrt[3]{x}$).

Running `testTask3()` in `StudentTester` should give the following output:

```
All wax dice before sorting:
[[B(Wax), V: 27.00, S: 54.00, M: 25,947.00, PE from 9m Height: 2,288,525.40, CanFloat: true, CanFly: false],
 [E(Wax), V: 8.00, S: 24.00, M: 7,688.00, PE from 9m Height: 678,081.60, CanFloat: true, CanFly: false],
 [D(Wax), V: 1.00, S: 6.00, M: 961.00, PE from 9m Height: 84,760.20, CanFloat: true, CanFly: false],
 [C(Wax), V: 8.00, S: 24.00, M: 7,688.00, PE from 9m Height: 678,081.60, CanFloat: true, CanFly: false],
 [A(Wax), V: 1.00, S: 6.00, M: 961.00, PE from 9m Height: 84,760.20, CanFloat: true, CanFly: false]]
All wax dice after sorting by volumes:
[[A(Wax), V: 1.00, S: 6.00, M: 961.00, PE from 9m Height: 84,760.20, CanFloat: true, CanFly: false],
 [D(Wax), V: 1.00, S: 6.00, M: 961.00, PE from 9m Height: 84,760.20, CanFloat: true, CanFly: false],
 [C(Wax), V: 8.00, S: 24.00, M: 7,688.00, PE from 9m Height: 678,081.60, CanFloat: true, CanFly: false],
 [E(Wax), V: 8.00, S: 24.00, M: 7,688.00, PE from 9m Height: 678,081.60, CanFloat: true, CanFly: false],
 [B(Wax), V: 27.00, S: 54.00, M: 25,947.00, PE from 9m Height: 2,288,525.40, CanFloat: true, CanFly: false]]

All butter balls before sorting:
[[Z(Butter), V: 4,442.92, S: 1,306.74, M: 4,047,503.91, PE from 9m Height: 356,989,844.84, CanFloat: true, CanFly: false],
 [X(Butter), V: 8,987.47, S: 2,090.11, M: 8,187,586.34, PE from 9m Height: 722,145,114.84, CanFloat: true, CanFly: false],
 [Y(Butter), V: 10,300.77, S: 2,289.06, M: 9,384,001.47, PE from 9m Height: 827,668,929.65, CanFloat: true, CanFly: false],
 [T(Butter), V: 4,442.92, S: 1,306.74, M: 4,047,503.91, PE from 9m Height: 356,989,844.84, CanFloat: true, CanFly: false],
 [S(Butter), V: 8,987.47, S: 2,090.11, M: 8,187,586.34, PE from 9m Height: 722,145,114.84, CanFloat: true, CanFly: false]]
All butter balls after sorting by surface areas:
[[T(Butter), V: 4,442.92, S: 1,306.74, M: 4,047,503.91, PE from 9m Height: 356,989,844.84, CanFloat: true, CanFly: false],
 [Z(Butter), V: 4,442.92, S: 1,306.74, M: 4,047,503.91, PE from 9m Height: 356,989,844.84, CanFloat: true, CanFly: false],
 [S(Butter), V: 8,987.47, S: 2,090.11, M: 8,187,586.34, PE from 9m Height: 722,145,114.84, CanFloat: true, CanFly: false],
 [X(Butter), V: 8,987.47, S: 2,090.11, M: 8,187,586.34, PE from 9m Height: 722,145,114.84, CanFloat: true, CanFly: false],
 [Y(Butter), V: 10,300.77, S: 2,289.06, M: 9,384,001.47, PE from 9m Height: 827,668,929.65, CanFloat: true, CanFly: false]]

Converting WaxDie B to a Sphere with radius: 1.86 and volume: 27.00
Converting ButterBall Z to a Cube with edge length: 16.44 and volume: 4,442.92
```

Note: compareTo(T o) returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object *o*.

Challenge Bonus (Optional):

In addition to the provided abstract class `Object3D` and interface `Melable`, create your own **abstract class** and **interface** that align with this lab's theme. Show how to use them.
